10-16-00

A

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No. **0100.0000810** Total Pages 31
First Inventor Andrzej Mamona et al.
Title Method and Apparatus for Constructing an Executable Program in Memory
Express Mail Label No. EL504284329US

APPLICATION ELEMENTS *See MPEP chapter 600 concerning utility patent application contents.*	*ADDRESS TO:* **Assistant Commissioner for Patents Box Patent Application Washington, DC 20231**

1. ☒ Fee Transmittal Form
(Submit an original, and a duplicate for fee processing)
2. ☒ Specification *Total Pages* 18
(preferred arrangement set forth below)
- Descriptive title of the Invention
- Cross References to Related Applications
- Statement Regarding Fed sponsored R & D
- Reference to Microfiche Appendix
- Background of the Invention
- Brief Summary of the Invention
- Brief Description of the Drawings *(if filed)*
- Detailed Description
- Claim(s)
- Abstract of the Disclosure

3. ☒ Drawings *(35 USC 113) Total Sheets* 6
4. Oath or Declaration *Total Pages* 2
a. ☒ Newly executed (original or copy)
b. ☐ Copy from a prior application (37 CFR 1.63(d))
(for continuation/divisional with Box 17 completed)
[Note Box 5 below]
i. ☐ DELETION OF INVENTOR(S)
Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).

5. ☐ Microfiche Computer Program *(Appendix)*

6. ☐ Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all necessary)*
a. ☐ Computer Readable Copy
b. ☐ Paper Copy (identical to computer copy)
c. ☐ Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. ☒ Assignment Papers (cover sheet & document(s))
8. ☒ 37 CFR 3.73(b) Statement (when there is an assignee) ☒ Power of Attorney
9. ☐ English Translation Document (if applicable)
10. ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations
11. ☐ Preliminary Amendment
12. ☒ Return Receipt Postcard (MPEP 503)
(Should be specifically itemized)
13. ☐ Small Entity Statement(s) ☐ Statement filed in Prior Application, Status still proper and desired.
14. ☐ Certified Copy of Priority Document(s)
(if foreign priority is claimed)
15. ☐ Other

16. If a CONTINUING APPLICATION, *check appropriate box and supply the requisite information:*
☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No:
Prior Application Information: Examiner *Group / Art Unit:*

17. CORRESPONDENCE ADDRESS

☐ *Customer Number or Bar Code Label* or, ☒ *Correspondence Address Below*

Markison & Reckamp, P.C.
P.O. Box 06229
Wacker Drive
Chicago, Illinois 60606-0229
Telephone: 312-939-9800 **Facsimile: 312-939-9828**

Name (Print/Type)	**Christopher J. Reckamp**	REGISTRATION NUMBER	**34,414**
Signature		Date	**Oct. 13, 2000**

FEE TRANSMITTAL

Note Effective October 1, 1997
Patent fees are subject to annual revision

TOTAL AMOUNT OF PAYMENT ($) 750.00

Complete if Known

Application Number	
Filing Date	**October 13, 2000**
First Named Inventor	**Andrzej Mamona**
Group Art Unit	
Examiner Name	
Attorney Docket Number	**0100.0000810**

METHOD OF PAYMENT (check one)

1. ☒ The Commissioner is hereby authorized to charge indicated fees and credit any over payments to

Deposit Account Number	50-0441
Deposit Account Name	**ATI Technologies, Inc.**

☒ Charge Any Additional Fee Required Under 37 CFR 1 16 and 1.17

☒ Charge the Issue Fee Set in 37 CFR 1 18 at the mailing of the Notice of Allowance

2. ☐ Payment Enclosed:
☐ Check ☐ Money Order ☐ Other

FEE CALCULATION

1. FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
101	690	201	345	Utility filing fee	**710.00**
106	310	206	155	Design filing fee	
107	480	207	240	Plant filing fee	
108	760	208	380	Reissue filing fee	
114	150	214	75	Provisional filing fee	

SUBTOTAL (1) ($) 710.00

2. CLAIMS

Claims	Extra	Fee from below	Fee Paid
Total 14	(-20 =)		
Indep.	(3 =)		
Multiple Dep.			

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
103	18	203	9	Claims in excess of 20
102	78	202	39	Independent claims in excess of 3
104	260	204	130	Multiple dependent claim
109	78	209	39	Reissue independent claims over original patent
110	18	210	9	Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($) _____

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
105	130	205	65	Surcharge - late filing fee or oath	
127	50	227	25	Surcharge - late provisional filing fee or cover sheet	
139	130	139	130	Non-English specification	
147	2,520	147	2,520	For filing a request for reexamination	
112	920*	112	920*	Requesting publication of SIR prior to Examiner action	
113	1,840*	113	1,840*	Requesting publication of SIR after Examiner action	
115	110	215	55	Extension for reply within first month	
116	380	216	190	Extension for reply within second month	
117	870	217	435	Extension for reply within third month	
118	1,360	218	680	Extension for reply within fourth month	
128	1,850	228	925	Extension for reply within fifth month	
119	300	219	150	Notice of Appeal	
120	300	220	150	Filing a brief in support of an appeal	
121	260	221	130	Request for oral hearing	
138	1,510	138	1,510	Petition to institute a public use proceeding	
140	110	240	55	Petition to revive - unavoidable	
141	1,210	241	605	Petition to revive - unintentional	
142	1,210	242	605	Utility issue fee (or reissue)	
143	430	243	215	Design issue fee	
144	580	244	290	Plant issue fee	
122	130	122	130	Petitions to the Commissioner	
123	50	123	50	Petitions related to provisional applications	
126	240	126	240	Submission of Information Disclosure Stmt	
581	40	581	40	Recording each patent assignment per property (times number of properties)	**40.00**
146	690	246	345	Filing a submission after final rejection (37 CFR 1 129(a))	
149	690	249	345	For each additional invention to be examined (37 CFR 1.129(b))	
Other fee (specify)					
Other fee (specify)					

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) ($) 40.00

SUBMITTED BY: MARKISON & RECKAMP, P.C.				Complete (if applicable)	
Typed or Printed Name	Christopher J. Reckamp			Reg. Number	34,414
Signature		Date	Oct. 13, 2000	Deposit Account User ID	50-0441

PATENT APPLICATION
DOCKET NO. 0100.0000810

In the United States Patent and Trademark Office

FILING OF A UNITED STATES PATENT APPLICATION

Title:

METHOD AND APPARATUS FOR CONSTRUCTING AN EXECUTABLE PROGRAM IN MEMORY

Inventors:

Andrzej Mamona **126 Pinedale Gate** **Woodbridge, Ontario, Canada**	**Indra Laksono** **138 Old Hill Street** **Richmond Hill, Ontario, Canada**

Attorney of Record
Christopher J. Reckamp
Registration No. 34,414
P.O. Box 06229
Wacker Drive
Chicago, Illinois 60606-0229
Phone (312) 939-9800
Fax (312) 939-9828

Express Mail Label No.
EL504284329US

Date of Deposit. 10/13/00

I hereby certify that this paper is being deposited with the U.S. Postal Service "Express Mail Post Office to Addresses" service under 37 C.F.R. Section 1.10 on the 'Date of Deposit', indicated above, and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Name of Depositor Rosalie Swanson
(print or type)

Signature. Rosalie Swanson

PATENT APPLICATION

0100.0000810

METHOD AND APPARATUS FOR CONSTRUCTING AN EXECUTABLE PROGRAM IN MEMORY

Field Of The Invention

The invention relates generally to methods and apparatus for constructing executable programs and more particularly to methods and apparatus for constructing executable programs, such as software drivers, in memory.

Background Of The Invention

Computers, Internet appliances, and other devices typically use software drivers that are executed by a host processor or peripheral processor, such as a graphics accelerator or any other suitable processing device. A driver, as used herein, includes, for example, a software driver that processes operating system requests or other suitable requests and programs hardware to perform operations. Typically, different software drivers are written so that they can be run by different system configurations. For example, computers may have differing manufacturers' central processing units with different hardware configurations such as system memory size, size and type of video memory, clock speed, and other system configurations. The software driver needs to decide the best path for its code to execute based on the type of CPU and hardware configuration that the driver is running on. This can help improve performance of the hardware for which the software driver is designed. Accordingly, it would be desirable to configure a software driver that can execute on a plurality of different devices so that differing software drivers need not be designed and stored for each different type of hardware configuration.

Typically, software drivers, such as display drivers and any other drivers, may be stored on a CD ROM and shipped to a customer. The CD ROM with the display driver,

or any other suitable driver, may include a plurality of different drivers that are used for different hardware configurations. For example, one driver may be provided for a hardware configuration that uses a first CPU type at a first operating speed along with a graphics accelerator that operates at a specified speed, and a different driver for use with a different type of CPU operating at a different speed for a different version of the same or different graphics accelerator system. Accordingly, a different software driver is used for different hardware configurations for an Internet appliance or computer.

A challenge arises to design a software driver that provides a suitable speed of executing operations so that high system performance is perceived from the perspective of an application and a user. For example, in the case of a display driver, a display driver executes operations for a graphics accelerator to perform three dimensional image rendering by a video game application, or any other suitable application. The software display driver should execute quickly and therefore typically the shorter the code path, the faster the operation of the driver. Accordingly, it is desirable to provide a minimum number of decisions that the code has to make with the actual hardware configuration. For example, the graphics drawing capabilities may be different for different controller chips and differing CPU capabilities, memory types, etc. Each graphics chip may typically have its own software driver. Software drivers may be application program interfaces (APIs) or other software modules that have entry points in the code that the operating system calls to perform functions. For example, in the case of the display driver, the operating system may call the display driver to have a graphics accelerator, also known as a graphics controller, draw a line for display on a display device. A GDI typically defines code entry points.

Also, software drivers typically evolve from one generation of a product to another wherein additional coding is added to previous code. If the older code has software fixes for hardware problems, and later versions of the hardware are fixed, there is no need to execute old or unnecessary code since this takes additional and valuable processing time.

Therefore, a need exists for an executable program, such as a software driver, that drives a plurality of peripherals and can run more efficiently on a plurality of CPUS's and host systems.

Brief Description Of The Drawings

The invention will be more readily understood with reference to the following drawings wherein:

FIG. 1 is a block diagram illustrating software components AND MEMORY used to construct an executable program in accordance with one embodiment of the invention.

FIG. 2 is a block diagram representing one example of software blocks used in constructing an executable program in memory in accordance with one embodiment of the invention.

FIG. 3 is a block diagram illustrating a code module library and a code bundle in accordance with one embodiment of the invention.

FIG. 4 is a flow chart illustrating one example of A method for constructing an executable program in memory in accordance with one embodiment of the invention;

FIG. 5 is a flow chart illustrating one example of a method for constructing an executable program in memory in accordance with one embodiment of the invention.

FIG. 6 is a graphic illustration of A mini-driver and jump instruction patching in accordance with one embodiment of the invention.

Detailed Description Of a Preferred Embodiment of The Invention

A method and apparatus for constructing an executable program, such as drivers, in memory includes obtaining actual dynamic and static system configuration parameters and dynamically constructing driver code bundles from a set of code modules obtained from a library, based on the actual system configuration parameters. The set of code modules includes code modules associated with a plurality of system configuration parameters. One example of the actual system configuration parameters includes static system configuration parameters such as in the case of a computer, a CPU type, clock speed and system memory size graphic accelerator type and associated video memory size. Other actual system configuration parameters include dynamic configuration

parameters which can be changed by the user or other application. One example of a dynamic configuration parameter may be, for example, pixel depth and display screen resolution. After obtaining optimal system configuration parameters depending upon a system's setting or configurations, dedicated code fragments such as code modules are used and stored in system memory or other suitable memory. Accordingly, optimal driver code is loaded at all times for a particular chip set and no unnecessary code is loaded from a CD ROM or other source. Since the code modules are selected based on a given set of configuration parameters an optimal driver is stored for a given system. The two types of actual system configuration parameters that determine a dynamic code bundle are the static, such as configuration parameters that never change during an operating session (hardware configuration) and dynamic configuration parameters that depend, for example, on software configuration (system flags, pixel depth, screen resolution, etc.). The dynamic configuration parameters and the static system configuration parameters are stored and are used as a type of index to determine a set of code modules that are selected to define a code bundle. A different code bundle may be stored in memory depending upon the different dynamic configuration parameter being selected.

In one embodiment, the method includes determining static system configuration parameters and storing static configuration parameters in driver memory (e.g., system memory where the driver is stored) for future use. The method also includes receiving a hardware enable call (through an enable call entry point) and obtaining dynamic configuration parameters associated with the hardware used by a software driver. The method also includes dynamically constructing code bundles for an actual system configuration based on the stored static and dynamic configuration parameters. The dynamic construction of code bundles is done for every entry point associated with a driver. For example, where the driver is a display driver, code bundles are dynamically constructed for each function associated with the display driver such as a draw circle function, draw line function, or any other defined function. The method also includes initializing the hardware, such as a graphics accelerator, by executing a constructed code bundle. The method also includes reconstructing a new code bundle after dynamic

configuration parameter changes are detected. This may occur, for example, when a change in display screen resolution is selected by a user.

FIG. 1 diagramatically illustrates construction of an executable program in accordance with one embodiment of the invention. A mini-driver 100, such as a dynamic load library or other suitable code module, contains jump instructions for each function associated with a given software driver. Driver entry points 102a-102n, as known in the art, include an entry point for each function to be performed by an executable program, such as the software driver. By way of example, and not limitation, the disclosed invention will be described with reference to a display driver used, for example, to control a graphics controller (graphics accelerator) that performs, as known in the art, image rendering including the drawing of graphics such as circles, lines, graphic overlays, graphic user interfaces and/or three dimensional rendering. An application, such as a video game or other suitable application, calls the software driver to perform drawing of images to be displayed on a display device. A dynamic code bundle, in one example, defines a portion of a display software driver capable of assisting in rendering graphics for display on a display device. However, the invention may be applicable to any suitable driver.

Sets of code modules, referred to herein as dynamic code modules 104, are stored disk files that store code modules to be loaded in dynamic code bundle construction memory 106 to form dynamic code bundles 116. A single library file, for example, of dynamic code modules is dedicated for service of a particular function that is carried out by the driver, for example, a Bit BLT operation, as known in the art of display drivers, or a draw circle function, or any other suitable function. Hardware initialization code 108 is used to initialize hardware such as a graphics controller or other suitable hardware that is used by the driver. The hardware initialization code 108 includes an entry point 110 referred to as an enable call entry point. A dynamic code module loader 112, in response to the enable call to the hardware initialization code 108, searches the dynamic code modules 104 (libraries) and dynamically selects code modules 114 based on stored static and dynamic configuration parameters. Dynamic code bundle construction memory 106

is system memory allocated by the dynamic code module loader 112 where dynamic code bundles are formed by the dynamic code modules being loaded, linked and executed. Accordingly, dynamic code bundles are constructed during execution of the enable function which enables hardware that is used by the driver. The display driver passes control to the dynamic code module loader 112. The dynamic code module loader 112 may be a software module.

The mini-driver 100 may be a 16-bit .dll or any other suitable size or type of software code. Each and every entry point 102a-102n associated with a given driver (i.e., functions of the driver) are patched with by a jump instruction. Accordingly, the mini-driver 100 includes a jump instruction for each entry point of a display driver, for example.

The dynamic code bundle 116 may be located in a 32-bit code segment (or any size code segment) and consists of parts of a driver's code which vary depending upon the configuration of the hardware associated with the system. There are multiple entry points to the dynamic code bundle 116. If desired, the dynamic code bundle 116 may be implemented as standard .dll chosen depending upon system settings and loaded using load library.

The dynamic code module loader 112 selects which dynamic code modules are to be used for a given driver function based on static system configuration parameters and dynamic system configuration parameters and loads the individual code modules into dynamic code bundle construction memory 106 and performs the suitable linking of each of the individual code modules to generate a dynamic code bundle that serves as a type of independent code segment that is executed to carry out a defined function of a driver such as a draw line function, draw circle function, Bit BLT function or any other suitable display driver or non-display driver function.

Referring to FIGs. 2-4, a dynamic code bundle database 200 contains an index corresponding to actual system configuration parameters wherein actual system

configuration parameters are associated with code modules stored in the dynamic code module library 104. For example, the dynamic code bundle database 200 has a collection of records describing relationships between code modules and a system configuration. Code modules are stored, on a per function basis, in the libraries in the dynamic code module library 104 as object modules. However, code modules may be defined in any other suitable way. Each of these code modules in a library has a certain record in the dynamic code bundle database. This record defines circumstances under which a particular code module should be loaded. Accordingly, the dynamic code bundle database 200 serves as an index database that contains code module descriptions by configuration parameters. For example, if a static system configuration indicated that the host CPU was an Intel Pentium®-based CPU with an optimized instruction set, a "CPU type" configuration parameter would be associated with a code module that was written in optimized instructions. For example, if a code module library was defined for the function of drawing a circle (see, e.g., FIG. 3), the code module library for drawing a circle may include a library of code modules that are written for differing CPU types and system memory size types. The dynamic code bundle database 200 includes a system parameter with associated code module identifiers to identify which code modules should be selected based on the particular configuration parameter. The Intel Pentium ®-III supports new instructions that allows faster 4-way floating point operations. These new instructions can be loaded and linked and then executed as needed on a Pentium-III without the need of a conditional test and branch if the host machine is indeed running a Pentium-III CPU. The saving of this extra compare saves not only the potential loss of time from branch misprediction in all the entry points, it also decreases the memory footprint of loaded drivers from having to load the new and older code fragments in memory. This saving applies to all the code paths that may benefit from these new instructions. There are many other areas of potential factors that will affect the choice of code path. Examples include but are not limited to BUS type (as certain buses allow faster access to graphics video memory and change the optimal path to load texture data), System Memory type (as slower system memory may change the way operations are assembled in memory), and VideoMemory size (as more or less video memory may

require some juggling of load balancing on which data stays in system memory and which is sent to video memory).

The dynamic code database may be defined a priori.

Referring to FIG. 4, a method for constructing executable programs, such as a driver, in system memory, is shown wherein an initial step includes an operating system booting to load the driver into memory, as shown in block 400. As shown in block 402, the driver initializes. As shown in block 404, the driver determines the static system configuration parameters of the system and stores the static actual system configuration parameter in driver memory for future use. This may be done, for example, by conventional methods, such as accessing an operating system file indicating the CPU type, memory size, and any other suitable static hardware system configuration information including the static system configuration associated with peripheral hardware, such as a graphics controller, video memory type, video memory quantity, graphics controller revision, system memory size and any other static system configuration parameters.

As shown in block 406, the driver passes control back to the operating system. As shown in block 408, the driver waits to receive a request from an application through the operating system such as a hardware enable. For example, the request may be to switch a display mode to change a screen resolution thereby changing dynamic system configurations. Accordingly, the pixel depth, for example, or other dynamic system configuration parameters, are then stored in driver memory, or any other suitable memory, for use by the dynamic code module loader to bundle the rest of the driver in a suitable manner. As shown in block 410, all dynamic configuration parameters are obtained so that at this point, the static and dynamic parameters are known and stored in driver memory. Accordingly, the driver obtains at least one actual system configuration parameter such as a static or dynamic system configuration parameter. Once the static and dynamic system configuration parameters are stored, the method includes dynamically constructing a driver code bundle for the actual system based on the dynamic and static configuration parameters, for every entry point associated with a

defined function. This is shown in block 412. For example, the clipping code can be defined with the two versions: Pentium III and non-Pentium III, and note that the Pentium III code is loaded for Pentium III or better. This can provide potential savings. There is a potential savings from just 3 points (a single triangle) as that makes the potential branch misprediction to test Pentium III-flag much higher as a percentage of total time.

Accordingly, the method includes dynamically constructing a code bundle from a set of code modules, such as a library 300 (FIG. 3) of code modules 302a-302n. As shown in FIG. 3, a set of code modules 300 includes code modules associated with system configuration parameters such as whether a CPU-type utilizes optimized instruction calculations, and another code module associated with a different system configuration parameter, such as selected pixel depth which may be, for example, a dynamic configuration parameter. A code bundle (where it is necessary) can be dynamically constructed for every affected driver's entry point associated with the software driver. Once the code bundle has been constructed by loading the requisite code modules in the dynamic code bundle construction memory 106 the dynamic code bundle loader links each of the individual code bundles so that the code bundle is a sequence of instructions with suitable address links. Note that code fragments are written to reference only relative addresses of code and data, accordingly, the code can be moved into any desired position, without having to do fixups of absolute code and data addresses, since they do not exist in the code fragments.

As shown in block 414, once the code bundles are constructed, the driver initializes the hardware and as shown in block 416, a new code bundle will only be generated after a detected change in dynamic configuration factors. If a change is detected in dynamic configuration parameters, the dynamic code module loader will reconstruct a new code bundle for that given function consistent with the changed configuration parameters. For example, if a user again changes the resolution

requirements for a display device, a new display function code bundle may be reconstructed.

Referring back to FIG. 2, the static hardware configuration may be stored in driver memory 202 and, if desired, a registry 204 may also be used to speed up the construction of a code bundle. For example, since code bundles are constructed during execution of an enable function, the driver passes control to the dynamic code module loader which recognizes the elements of the actual system environment by evaluating the stored static and dynamic configuration parameters. The registry 204 is updated during the user installation of a driver set. Alternately, a configuration optimizer can be optionally run by the user on each change to determine the new optimal registry configuration. The Registry 204 contains flags that define the optimal settings for what modules to load. The only flags that are typically used are the ones that affect optimal code execution. It does not contain critical flags for CPU type or any flag that could cause invalid code to be loaded (eg : PIII code loaded on PII CPU). The critical flags are determined on each bootup of the driver. If a registry search is successful, the dynamic code module loader simply configures the dynamic code bundle memory with code modules according to the list stored in the registry. Otherwise, it is necessary to review the dynamic code bundle database and store settings in the registry. For example, – for a given configuration, DCB settings are always the same, so indices are stored in a data base. It will be recognized that any other information that describes how bundles should be constructed can also be used. When an optimal setting cannot be determined, a general purpose non-optimal fallback code bundle can be loaded. This code bundle may contain all the tests and check to run from the fastest to the slowest systems. Alternately, it can simply contain the slowest code that works on all systems.

As shown in FIG. 3, each code bundle 116 includes a plurality of selected code modules 114. In response to storing dynamic configuration parameters, the driver, namely the dynamic code module loader, uses indexed code modules associated with the stored dynamic configuration parameters as defined in the dynamic code bundle database

200 to determine which code modules are selected to define a portion of a dynamic code bundle of the software driver.

FIG. 5 illustrates one example of the operation of the dynamic code module loader 112. As shown in block 500, the dynamic code module loader analyzes the stored static and dynamic configuration parameters 202 and 203. As shown in block 502, the dynamic code module loader determines from the dynamic code bundle database what code modules are to be used for the actual system configuration based on the stored dynamic and static configuration parameters.

The dynamic code module library 104 containing the plurality of code module libraries may be stored, for example, on a system's hard drive, if desired. In addition, the dynamic code bundle construction memory is preferably stored on a system hard drive. Typcially, libraries of code bundles from removable medias like CDs or floppy disks may be stored to the hard drive, so these files would be accessible always when reconstruction of bundles is required. However it wil be recognized that source libraries may be obtained from other media including storage available via the Internet.

As shown in block 504, the dynamic code module loader moves a selected code module into the code bundle construction memory to form a dynamic code bundle. As shown in block 506, the dynamic code module loader performs linking between selected code modules in the construction memory. This occurs typically since a jump instruction that is included as part of an original code module may have the wrong address. Also, a variable included in the code module in the library may have wrong data and this information needs to be changed when it is linked with other code modules. As shown in block 506, once the linking is complete, the dynamic code module loader patches a jump instruction in the mini-driver for entry points associated with the given function so that when the function is called, it jumps to the appropriate dynamic code bundle stored in the dynamic code bundle construction memory. Linking is a process of resolving external references between boundaries. During this process symbolic information, such

as names of variables and/or labels in a code are being replaced by addresses of these objects, so the program can be executed

FIG. 6 illustrates diagramatically the patched jump instructions 600a-600n which define the mini-driver 100 associated with the display driver. There is a jump instruction associated with each dynamic code bundle. Each dynamic code bundle, when executed, performs a given driver function, such as a draw circle function, draw line function or any other suitable function. Each jump instruction causes the processing device to jump to the appropriate dynamic code bundle which is then executed. A GDI, as known in the art, generates the calls 602a-602n to the entry points of the mini-driver 100 In the above example, a single minidriver with multiple entry points (functions) is used, but the method is applicable to any software module, and not necessarily a driver.

The above software may be stored, for example, on one or more storage mediums that contain executable instructions that when executed by one or more processing devices, causes the one or more processing devices to operate as described above. The storage medium may include a CD ROM, device (system) memory, RAM, server memory or any suitable memory. The disclosed driver architecture provides for only required software modules to be executed, thereby bypassing older sections of software that may no longer be applicable or software fixes that are no longer required to run in view of an improved version of hardware that has been detected through the configuration parameters. The dynamic code bundles are preferably but not necessarily loaded into RAM of the system. Other advantages will be recognized by those of ordinary skill in the art.

It should be understood that the implementation of other variations and modifications of the invention in its various aspects will be apparent to those of ordinary skill in the art, and that the invention is not limited by the specific embodiments described. It is therefore contemplated to cover by the present invention, any and all modifications, variations, or equivalents that fall within the spirit and scope of the basic underlying principles disclosed and claimed herein.

Claims

WHAT IS CLAIMED IS:

1. A method for constructing an executable program in memory comprising the steps of:

obtaining at least one actual system configuration parameter; and

dynamically constructing at least one code bundle from a set of code modules based on the actual system configuration parameter,

wherein the set of code modules includes at least one code module associated with a first system configuration parameter and at least a second code module associated with a second system configuration parameter.

2. The method of claim 1 wherein the executable program is associated with a software driver and including the step of dynamically constructing a code bundle for every driver entry point associated with the software driver.

3. The method of claim 1 wherein the step of dynamically constructing the at least one code bundle includes adding at least one of a a jump instruction and call instruction for every code bundle.

4. The method of claim 1 wherein each code bundle includes a plurality of code modules.

5. The method of claim 4 wherein the at least one actual system configuration parameter includes at least one of a dynamic configuration parameter and a static system configuration parameter and wherein the method includes the step of storing the dynamic configuration parameter or the static system configuration parameter.

6.The method of claim 5 including the step of storing a library containing at least the set of code modules and a storing a database containing at least an index corresponding to actual system configuration parameters wherein actual system configuration parameters are associated with at least one code module stored in the library and wherein the step of dynamically constructing at least one code bundle includes:

in response to storing dynamic configuration parameters, using indexed code modules associated with the stored dynamic configuration parameters to determine which code modules are selected to define a portion of a software driver.

7.The method of claim 1 wherein the dynamic code bundle defines at least a portion of a display software driver capable of assisting in rendering graphics for display on a display device.

8. A storage device that contains programming instructions that when executed by one or more processing devices causes the one or more processing devices to:

obtain at least one actual system configuration parameter; and

dynamically construct at least one code bundle from a set of code modules based on the actual system configuration parameter,

wherein the set of code modules includes at least one code module associated with a first system configuration parameter and at least a second code module associated with a second system configuration parameter.

9. The storage device of claim 8 including stored programming instructions that when executed by one or more processing devices causes the one or more processing devices to dynamically construct a code bundle for every driver entry point associated with a software driver.

10.The storage device of claim 8 including stored programming instructions that when executed by one or more processing devices causes the one or more processing devices to dynamically construct the at least one code bundle by adding a jump instruction for every code bundle.

11.The storage device of claim 8 wherein each code bundle includes a plurality of code modules.

12. 12. The storage device of claim 11 wherein the at least one actual system configuration parameter includes at least one dynamic configuration parameter and at least one static system configuration parameter and wherein the storage device includes stored programming instructions that when executed by one or more processing devices causes the one or more processing devices to store the dynamic configuration parameter and the static system configuration parameter.

13.The storage device of claim 12 including stored programming instructions that when executed by one or more processing devices causes the one or more processing devices to

store a library containing at least the set of code modules and store a database containing at least an index corresponding to actual system configuration parameters wherein actual system configuration parameters are associated with at least one code module stored in the library and in response to storing dynamic configuration parameters, using indexed code modules associated with the stored dynamic configuration parameters to determine which code modules are selected to define a portion of a software driver.

14. The storage device of claim 8 wherein the dynamic code bundle defines at least a portion of a display software driver capable of assisting in rendering graphics for display on a display device.

METHOD AND APPARATUS FOR CONSTRUCTING AN EXECUTABLE PROGRAM IN MEMORY

Abstract Of The Invention

A method and apparatus for constructing an executable program, such as drivers in memory, obtains system configuration parameters and dynamically constructs driver code bundles from a set of code modules obtained from a library, based on the actual system configuration parameters. The set of code modules includes code modules associated with a plurality of system configuration parameters. One example of the system configuration parameter include static system configuration parameters such as in the case of a computer, a CPU type, clock speed and system memory size. Other actual system configuration parameters include dynamic configuration parameters which can be changed by the user. One example of a dynamic configuration parameter may be, for example, pixel depth and display screen resolution. After obtaining optimal system configuration depending upon a system's setting or configurations, dedicated code modules are used and stored in system memory or other suitable memory. Accordingly, optimal driver code is loaded at all times for a particular chip set and no unnecessary code is loaded from a CD ROM or other source.

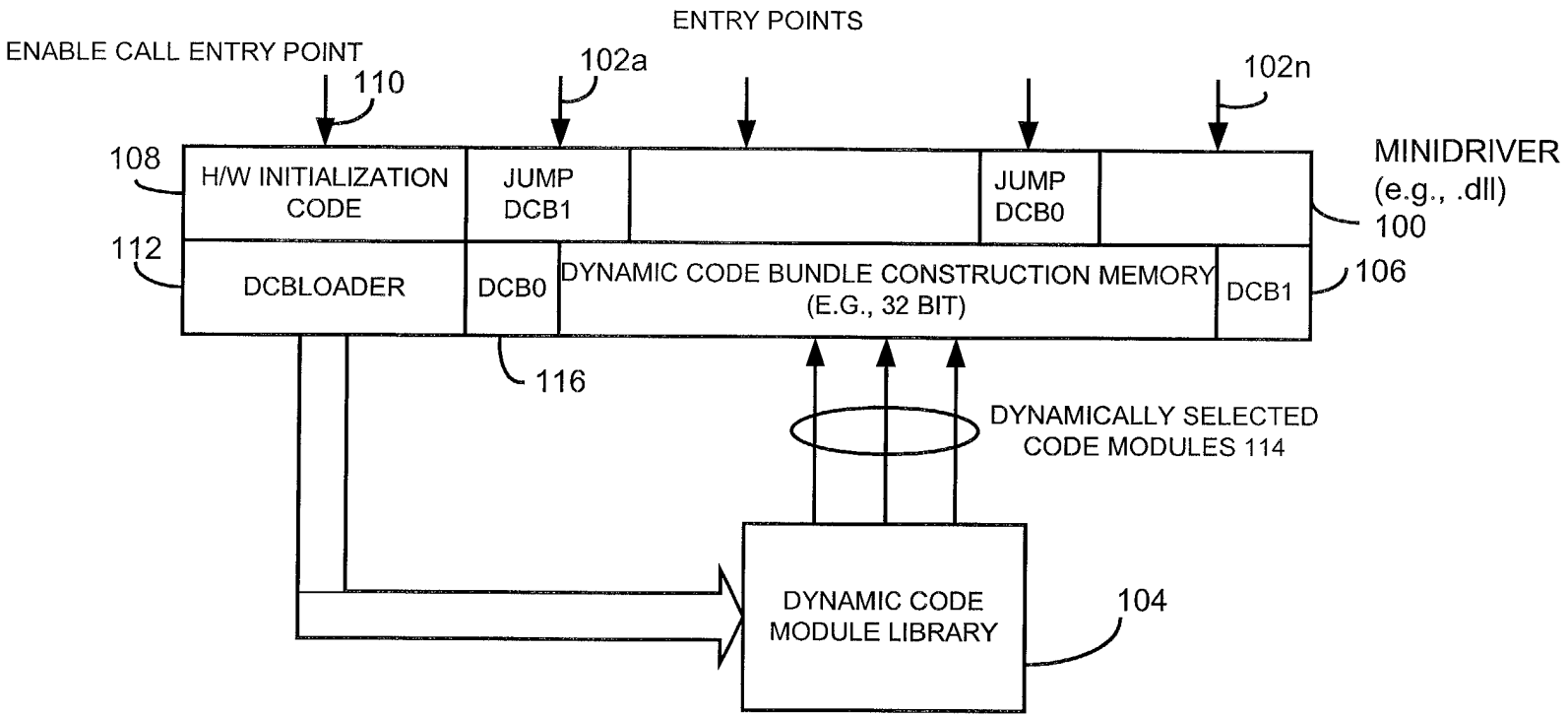

FIG. 1

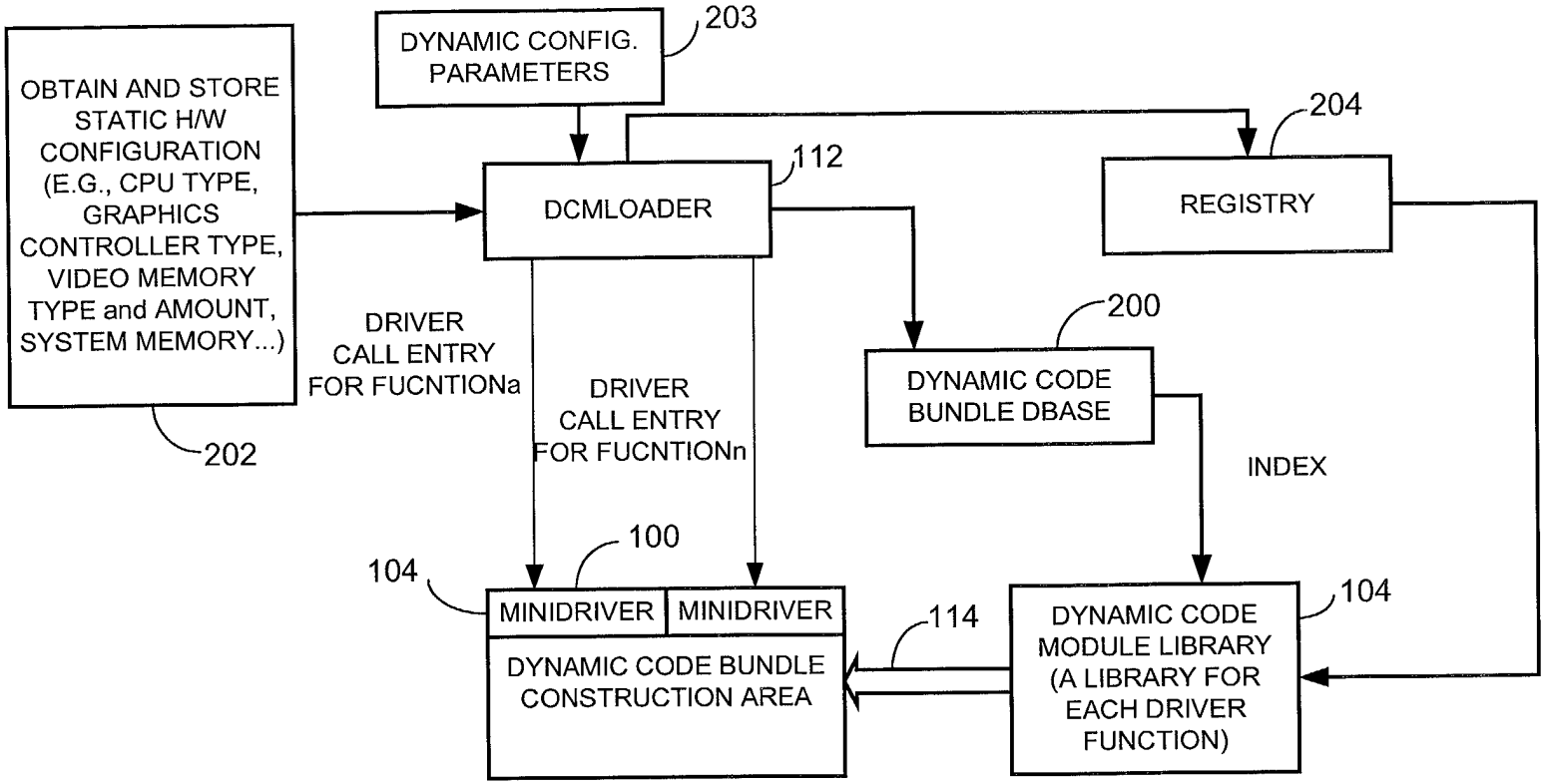
OBTAIN AND STORE STATIC H/W CONFIGURATION (E.G., CPU TYPE, GRAPHICS CONTROLLER TYPE, VIDEO MEMORY TYPE and AMOUNT, SYSTEM MEMORY...)
202
DYNAMIC CONFIG. PARAMETERS
203
DCMLOADER
112
REGISTRY
204
DRIVER CALL ENTRY FOR FUCNTIONa
DRIVER CALL ENTRY FOR FUCNTIONn
DYNAMIC CODE BUNDLE DBASE
200
INDEX
104
100
MINIDRIVER
MINIDRIVER
DYNAMIC CODE BUNDLE CONSTRUCTION AREA
114
DYNAMIC CODE MODULE LIBRARY (A LIBRARY FOR EACH DRIVER FUNCTION)
104

FIG. 2

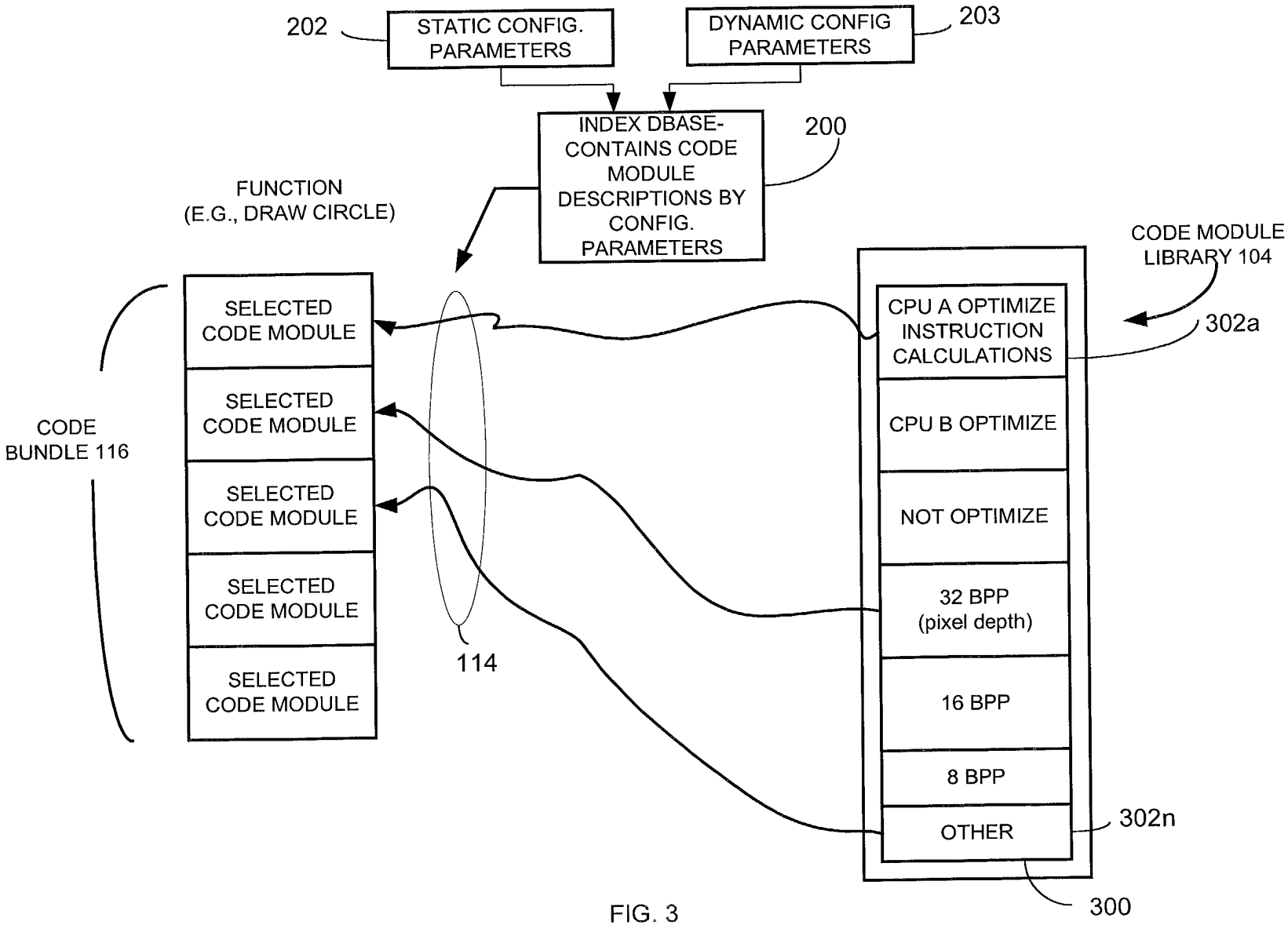
202
STATIC CONFIG. PARAMETERS
DYNAMIC CONFIG PARAMETERS
203
INDEX DBASE- CONTAINS CODE MODULE DESCRIPTIONS BY CONFIG. PARAMETERS
200
FUNCTION (E.G., DRAW CIRCLE)
CODE MODULE LIBRARY 104
SELECTED CODE MODULE
SELECTED CODE MODULE
SELECTED CODE MODULE
SELECTED CODE MODULE
SELECTED CODE MODULE
CODE BUNDLE 116
114
CPU A OPTIMIZE INSTRUCTION CALCULATIONS
302a
CPU B OPTIMIZE
NOT OPTIMIZE
32 BPP (pixel depth)
16 BPP
8 BPP
OTHER
302n
300

FIG. 3

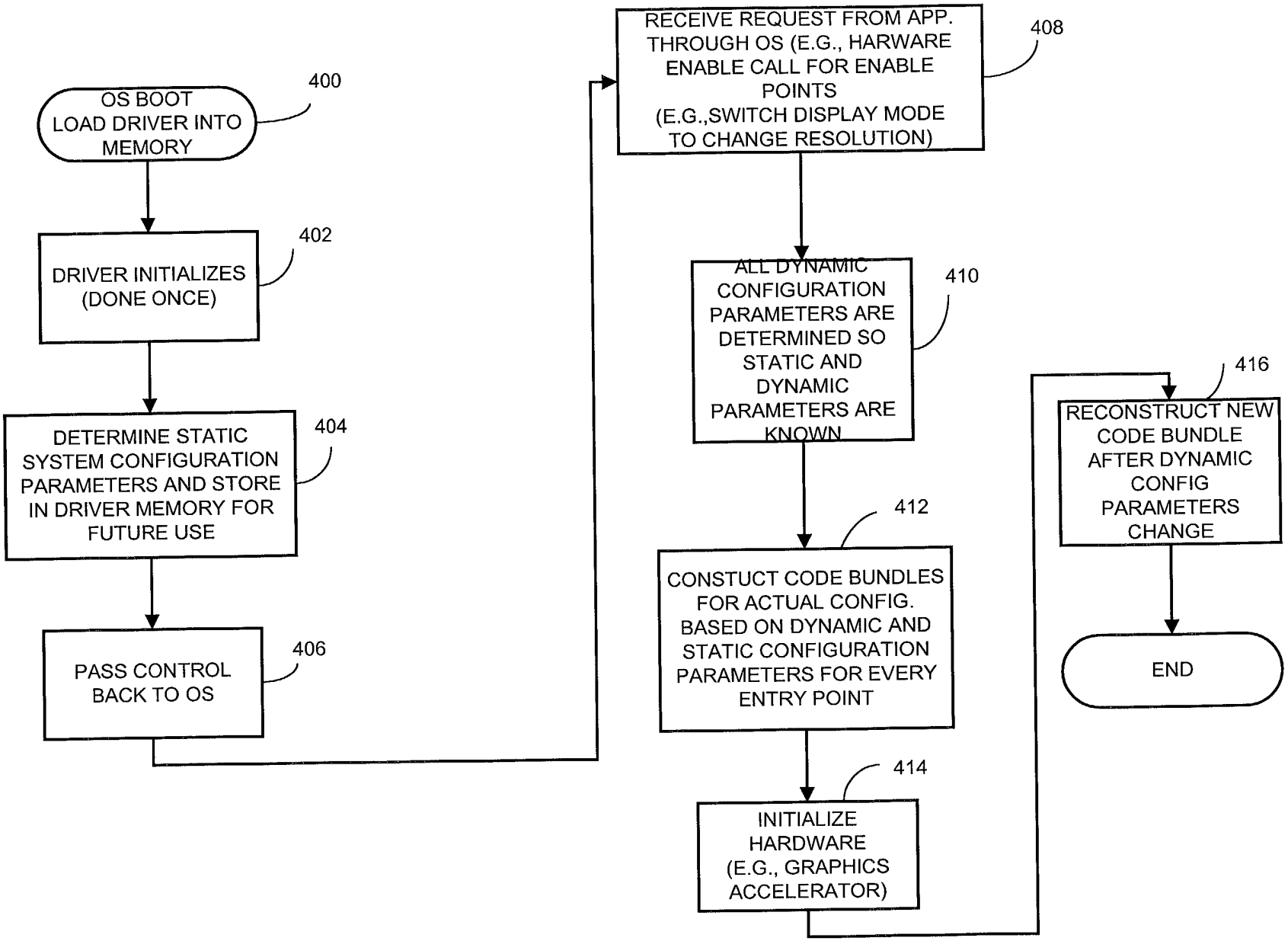
OS BOOT LOAD DRIVER INTO MEMORY
400
DRIVER INITIALIZES (DONE ONCE)
402
DETERMINE STATIC SYSTEM CONFIGURATION PARAMETERS AND STORE IN DRIVER MEMORY FOR FUTURE USE
404
PASS CONTROL BACK TO OS
406
RECEIVE REQUEST FROM APP. THROUGH OS (E.G., HARWARE ENABLE CALL FOR ENABLE POINTS (E.G.,SWITCH DISPLAY MODE TO CHANGE RESOLUTION)
408
ALL DYNAMIC CONFIGURATION PARAMETERS ARE DETERMINED SO STATIC AND DYNAMIC PARAMETERS ARE KNOWN
410
CONSTUCT CODE BUNDLES FOR ACTUAL CONFIG. BASED ON DYNAMIC AND STATIC CONFIGURATION PARAMETERS FOR EVERY ENTRY POINT
412
INITIALIZE HARDWARE (E.G., GRAPHICS ACCELERATOR)
414
RECONSTRUCT NEW CODE BUNDLE AFTER DYNAMIC CONFIG PARAMETERS CHANGE
416
END

FIG. 4

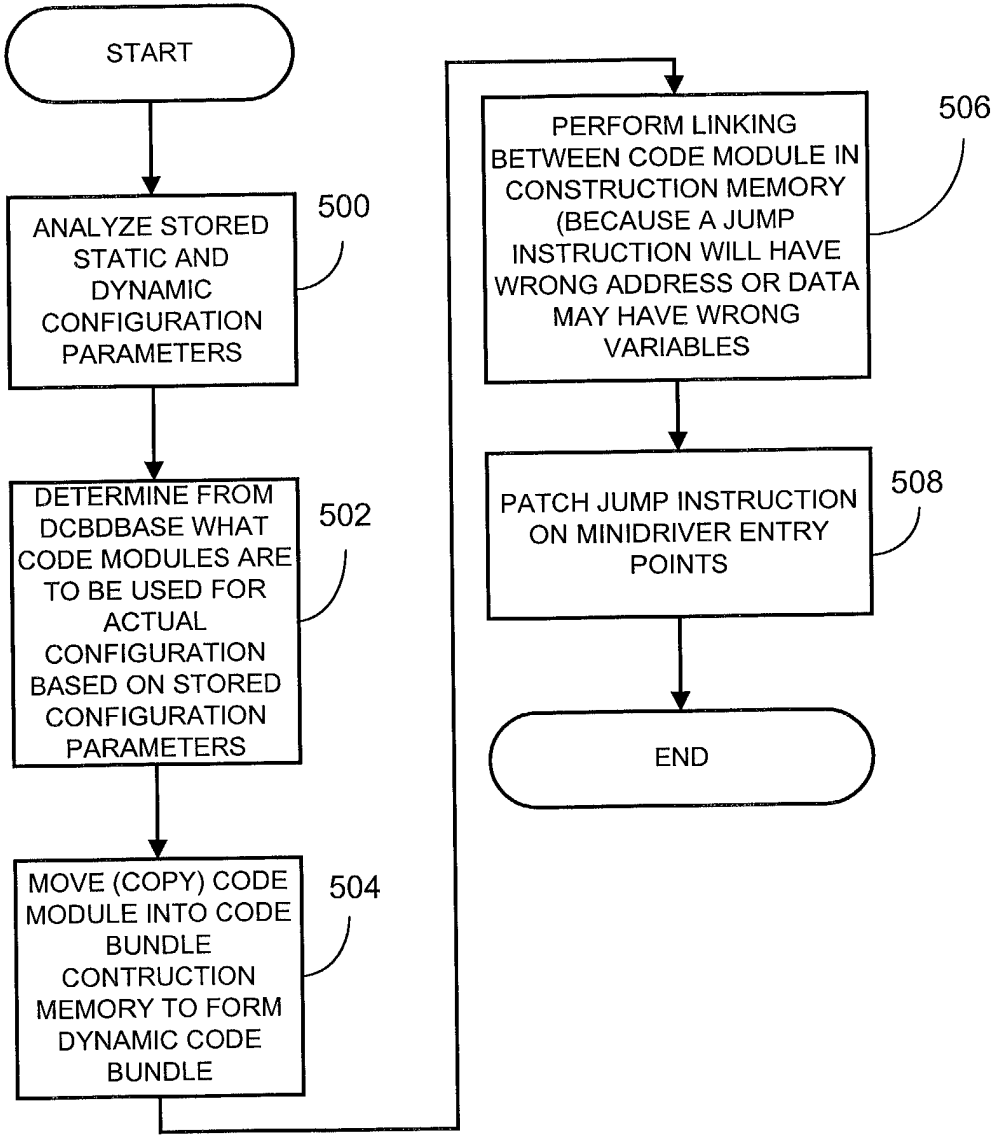
START
ANALYZE STORED STATIC AND DYNAMIC CONFIGURATION PARAMETERS
500
DETERMINE FROM DCBDBASE WHAT CODE MODULES ARE TO BE USED FOR ACTUAL CONFIGURATION BASED ON STORED CONFIGURATION PARAMETERS
502
MOVE (COPY) CODE MODULE INTO CODE BUNDLE CONTRUCTION MEMORY TO FORM DYNAMIC CODE BUNDLE
504
PERFORM LINKING BETWEEN CODE MODULE IN CONSTRUCTION MEMORY (BECAUSE A JUMP INSTRUCTION WILL HAVE WRONG ADDRESS OR DATA MAY HAVE WRONG VARIABLES
506
PATCH JUMP INSTRUCTION ON MINIDRIVER ENTRY POINTS
508
END

FIG. 5

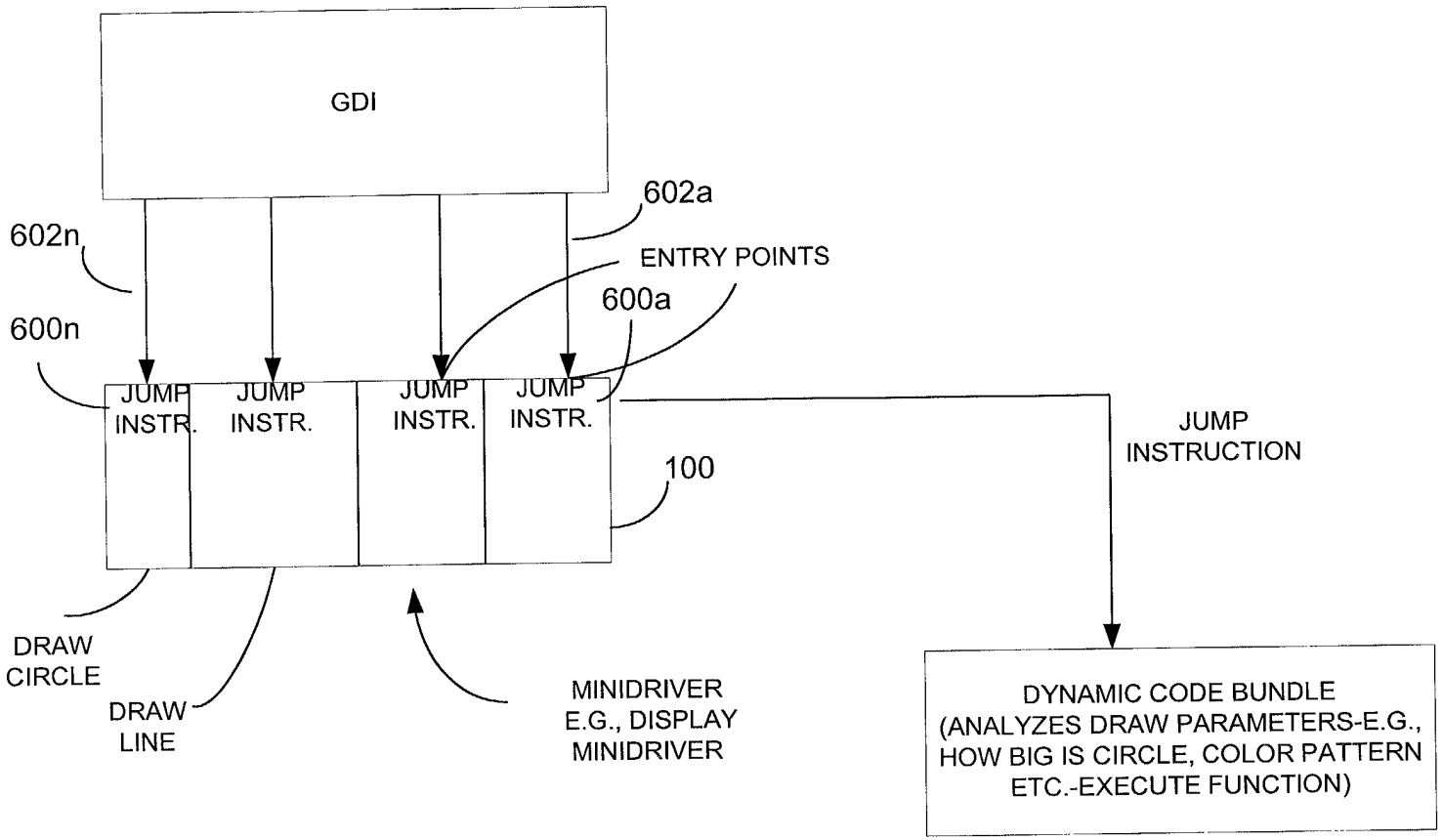
GDI
602a
602n
ENTRY POINTS
600a
600n
JUMP INSTR.
JUMP INSTR.
JUMP INSTR.
JUMP INSTR.
100
JUMP INSTRUCTION
DRAW CIRCLE
DRAW LINE
MINIDRIVER E.G., DISPLAY MINIDRIVER
DYNAMIC CODE BUNDLE (ANALYZES DRAW PARAMETERS-E.G., HOW BIG IS CIRCLE, COLOR PATTERN ETC.-EXECUTE FUNCTION)

FIG. 6

DECLARATION FOR UTILITY OR DESIGN PATENT APPLICATION (37 CFR 1.63)

☒ Declaration Submitted with Initial Filing, OR
☐ Declaration Submitted after Initial Filing
(surcharge (37 CFR 1.16 (e)) required)

Attorney Docket Number 0100.0000810
First Named Inventor A. Mamona et al.
COMPLETE IF KNOWN
Application Number
Filing Date
Group Art Unit
Examiner Name

As a below named inventor, I hereby declare that:
My residence, post office address, and citizenship are as stated below next to my name.
I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: METHOD AND APPARATUS FOR CONSTRUCTING AN EXECUTABLE PROGRAM IN MEMORY
the specification of which:
☒ is attached hereto.
☐ was file on (MM/DD/YYYY) as United States Application Number or PCT International Application Number and was amended on (MM/DD/YYYY) (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.
I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached? YES	NO
			☐	☐	☐
			☐	☐	☐

☐ Additional foreign application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

I hereby claim the benefit under 35 U.S.C. 119(e) of any United States provisional application(s) listed below.

Application Number(s)	Filing Data (MM/DD/YYYY)

☐ Additional provisional application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

U.S. Parent Application or PCT Parent Number	Parent Filing Date (MM/DD/YYYY)	Parent Patent Number (if applicable)

☐ Additional U.S. or PCT international application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

As a named inventor, I hereby appoint the following registered practitioner(s) to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Name	Registration Number	Name	Registration Number
John R. Garrett	27,888	Christopher J. Reckamp	34,414
Daniel C. Crilly	38,417		
Sally Daub	41,478		

☐ Additional registered practitioner(s) named on supplemental Registered Practitioner Information sheet PTO/SB/02C attached hereto.

Direct all correspondence to:

Markison & Reckamp, P.C.
P.O. Box 06229
Wacker Drive
Chicago, Illinois 60606-0229
Telephone:312-939-9800
Facsimile: 312-939-9828

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Name of Sole or First Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])	Family Name or Surname		
Andrzej	Mamona		
Inventor's Signature		Date	Oct 10, 2000
Residence City: Woodbridge	State: Ontario	Country: Canada	Citizenship: Canada
Post Office Address	126 Pinedale Gate		
City: Woodbridge	State: Ontario	ZIP: L4L 8X3	Country: Canada

Name of Additional Joint Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])	Family Name or Surname		
Indra	Laksono		
Inventor's Signature		Date	October 12, 2000
Residence City: Richmond Hill	State: Ontario	Country: Canada	Citizenship: Canada
Post Office Address	138 Old Hill Street		
City: Richmond Hill	State: Ontario	ZIP: L4C 9Z7	Country: Canada

Name of Additional Joint Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])	Family Name or Surname		
Inventor's Signature		Date	
Residence City:	State:	Country:	Citizenship:
Post Office Address			
City:	State:	ZIP:	Country:

☐ Additional inventors are being named on the _____supplemental Additional Inventor(s) sheet(s) PTO/SB/02A attached hereto.